CG2271 Real Time Operating system AY 2017

Lab 6 – Automated Driving System (10% of your final grade, 70 marks)

Lab Lessons: Wednesday 5 April & 12 April
IVLE Submission of Report and Code: Wednesday 19 April midnight
Demo: Wednesday 12 April or 19 April

Introduction:

The objective of this lab is for you to apply all the knowledge and the skills that you have developed in real-time operating systems towards solving a single, integrated, real-life problem. In particular, the lab explores embedded software design with focus on task management, real-time scheduling, interrupt handling, and task synchronization plus communication.

Your task is to design the controller of a car with an automated driving system. The software running on the Arduino platform will act as the high-level controller of the car. The interface with the real world will be emulated using different peripherals included in the Sparkfun kit. The main role of the controller is to maintain the desired speed of the car based on the driver input and brake automatically to avoid accidents if the vehicle in front is too close.

Sensors/Actuators:

The following table shows the sensors/actuators that you need to use in addition to the Arduino board.

	Quantity	Remarks
Buzzer/Motor	x1	Emulates the engine behavior according to the speed
Red LED	x1	Indicates that the automated safety brake is activated
Yellow LED	x3	Indicate the speed through the number of LEDs that are switched on (0 LED ON for the lowest speed, 3 LEDs ON for the highest speed)
Potentiometer	x1	Emulates the distance from the vehicle in front
Push Button	X2	Emulate the driver inputs to increase or decrease speed
UART Terminal	X1	Display the current/desired speed and the distance from vehicle in front

In terms of software, you need to set up FreeRTOS and enable semaphore options.

Required Functionalities:

- ***The car should support four discrete speed settings: 0 (lowest speed), 1, 2, 3 (highest speed).***

You will emulate the behavior of the engine at any speed by a buzzer, which is available in the Sparkfun kit. The buzzer should emit sounds at four different volume levels (soft to loud) or frequency levels (pitch) according to the four discrete speeds. You can find further instructions on how to use the buzzer in the Sparkfun manual provided with the kit. You may also refer to

https://cdn.sparkfun.com/datasheets/Kits/SFE03-0012-SIK.Guide-300dpi-01.pdf https://learn.sparkfun.com/tutorials/sik-experiment-guide-for-arduino---v32/experiment-11-using-a-piezo-buzzer

https://www.arduino.cc/en/Tutorial/PlayMelody

For your own interest (i.e., optionally), you may also emulate the behavior of the wheels using the small motor available in the Sparkfun kit. You can adjust the motor speed according to the current speed setting. But **using the motor is not a requirement** for full marks. If you decide to use the motor, please note that the motor cannot be driven directly from an output of the Arduino board. Instead, you have to use a transistor. You can find an example in the Sparkfun manual (https://learn.sparkfun.com/tutorials/sik-experiment-guide-for-arduino---v32/experiment-12-driving-a-motor).

Furthermore, the current speed of the car should be indicated by three yellow LEDs. The number of LEDs that should be turned ON is equal to the current speed value. In other words, no LED should be turned on at speed setting=0 and all the three LEDs should be turned ON at speed setting = 3.

- ***The driver should be able to change the speed of the car.***

The driver of the car should be able to change the speed of the car using the two push buttons. One push button acts as the accelerator (gas pedal) and the other acts as the brake. Initially, the car runs at speed setting = 0. When the accelerator push button is pressed, the driver speed setting increases by one discrete step (for example, 0 to 1). The only exception is the case where the current speed setting is equal to 3; in this case, the speed does not change. Similarly, when the brake push button is pressed, the driver speed setting decreases by one discrete step (for example, 2 to 1). The only exception is the case where the current speed setting is equal to 0; in this case, the speed does not change.

We assume the existence of a low-level controller that adjusts and maintains the speed requested by our high-level controller. The functionality of this low-level controller is outside the scope of this lab. You may simply assume that the engine changes speed according to the discrete desired speed settings, and you only need to select the desired speed setting.

- ***The car should be able to measure the distance from the vehicle in the front.***

You will use a potentiometer to emulate the behavior of the sensor(s) mounted at the front of the car providing distance information from the closest vehicle. You will have to divide the range of possible readings from the potentiometer into four parts. These parts will correspond to a distance of `d, 2d,` 3d, and 4d, respectively, from the vehicle in the front. The controller has to read the potentiometer input at least once every **500ms**.

- ***The car should support automated braking.***

The car controller should support an automated safety feature. If the vehicle in front is too close, the controller should decrease the current speed automatically to avoid accident. In this emergency (car in front is too close), the controller can override the driver input (driver speed setting) and the driver cannot increase the speed if it is limited by the distance reading. Once the distance from the vehicle in front increases beyond the required threshold, the controller goes back to setting the speed of the car according to the driver input.

The threshold distance (safe distance from the vehicle in front) for engaging the automated breaking depends on the current speed of the car. The higher the current speed of the car, the longer is the safe distance (because it will take shorter time for the car to hit the vehicle in front). The following table provides the safe distance values at different current speed.

Speed Setting	**Safe Distance**
0	d
1	2d
2	3d
3	4d

For example, suppose that the driver has set the desired speed setting to 2. However, suddenly the vehicle in the front slows down and the distance measured from the vehicle in front becomes equal to d. The safe distance at speed 2 is 3d. Therefore, the controller should reduce the current speed of the car to 0. Note that the automated braking system can change the current speed to any value in a single step (e.g., from 2 to 0) and you do not need to take into consideration the time required to slow down the car. As the current speed of the car decreases, the distance from the car is front is expected to increase (you should emulate this behavior by turning the potentiometer). As the distance from the vehicle in front increases, the controller can accordingly increase the current speed until the car can again run at the desired speed setting of the driver. In other words,

```
current speed = minimum (driver desired speed setting,
                          safe speed setting at current distance)
```

To indicate that the automated safety brake is activated/engaged, a red LED should be turned on for a period of **1-1.5 seconds**. Both the buzzer and the three yellow LEDs should indicate the new, reduced speed. If you are using the motor, it should indicate the new speed as well.

- ***The car should display status information through the UART terminal.***

You should display status information about the car through the UART terminal. At the minimum, you should display the driver desired speed setting, the current speed (different from the driver speed setting if the automated safety brake is engaged), and the distance reading from the vehicle in front. These information should be updated at least **once every second**.

<u>RTOS Task Requirements:</u>

You can implement the required functionality of the controller by choosing an appropriate set of tasks, their periods and priorities, interrupt functions, and the communication/synchronization among the tasks (or between interrupts and tasks). The exact choice of tasks, periods, and priorities are left to you. However, your implementation should satisfy the following constraints:

- You are required to use at least three RTOS tasks and two message queues. For example, you may use a task to manage the buzzer, a task to periodically read the potentiometer value (distance), a task to compute the current speed based on the driver speed setting and the distance reading.
- You should detect the pressing of the push buttons (for speed change by the driver) through interrupts.
- Use a separate RTOS task for the UART communication, and use a message queue for sending information to the communication task. Serial communication is slow; but it should not affect more critical functionalities of the controller (such as automatic braking). You should assign task priorities accordingly.
- You should not create and delete tasks repeatedly. All the tasks should be created in the beginning and execute either periodically or when needed.

Deliverables

Your deliverables include a report and the source code. You should upload the full source code including informative and necessary comments. Please upload report and source code to IVLE Files → Lab6-Submissions → Your Lab Group by Wednesday 19 April midnight.

Your report should include the following:

- A schematic of the circuit. You may use the Fritzing software or scan a hand-drawn figure; but make sure that your schematic is easy to read.
- The description of your solution. It should include the software architecture, the chosen ranges for potentiometer reading, the role of different tasks and interrupts, the periods and priorities of the tasks, how the communications among the tasks are achieved, and the scheduling policy.

Demo Session

You should demo your car controller to your TA on either Wednesday 12 April or Wednesday 19 April (it is up to you to decide the date). Please be punctual for your demonstration. Your demonstration is worth 10 marks.

Grading Scheme (Total 70 Marks)

- Correct implementation of buzzer: 5 marks
- Correct implementation of push buttons: 5 marks
- Correct implementation of potentiometer: 5 marks
- Correct implementation of Yellow LEDs: 5 marks
- Correct implementation of red LED: 5 marks
- Correct implementation of UART: 5 marks

- Reasonable choice of tasks: 3 marks
- Reasonable choice of task periods: 3 marks
- Reasonable choice of task priorities: 3 marks

- Implementation of interrupts: 3 marks
- Implementation of message queues: 3 marks

- Correct Circuit Schematic: 5 marks
- Demo: 10 marks
- Overall quality of your otherwise correct solution: 10 marks [This part evaluates the overall software architecture design, minimization of CPU utilization to leave resources for future functionality etc.]